38

Popular Computing

May 1976 Volume 4 Number 5

The Peripatetic Jumping Bean

12	23	8	10	5
4	25	3 ★★	20 ★	16
11	17	19	12	23
6	15	24	2	13
14	18	20	9	17

In the 5 x 5 array shown, each cell contains a random integer in the range from 1 to N. A jumping bean starts at the center cell and jumps according to this pattern:

1	2	3
4	X	5
6	7	0

where X is the number in the cell from which the bean jumps, and he goes to one of the 8 surrounding cells according to X mod 8.

Thus, with the contents of the cells as shown, the bean will move to the cell marked with the star (and then to the cell marked with two stars, and so on). The number in the cell the bean leaves is reduced by one. If a move cannot be made (because of the boundaries of the pattern), further attempts are made with the new reduced numbers. The pattern shown will appear like this after 5 moves:

12	23	08	07	05
04	25	02	19	14
11	17	18	12	23
06	15	24	02	13
14	18	20	09	17

The journey stops when it is not possible to move (for example, with the bean in any of the cells in the bottom row or the right column and the contents of that cell zero). The number of jumps made is counted (and notice that a jump is not counted while the bean marks time when the cell count goes down to make a jump possible) and called M.

The Problem is to find the average ratio of M/N and determine whether this ratio is in any way dependent on N or the size of the array. The average ratio for a given N will result from playing the game many times, holding N and the array size constant, but selecting different values for the cell contents at random.

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in the United States is $18 per year, or $15 if remittance accompanies the order. For Canada and Mexico, add $4 per year to the above rates. For all other countries, add $6 per year to the above rates. Back issues $2 each.

Pocket Calculator Review: SR-52

The SR-52 by Texas Instruments is the next logical advance in the evolution of pocket calculators. This programmable machine weighs 12 ounces (340 grams) and measures 3.125 x 6.5 x 1.75 inches (8 x 16.5 x 4.5 cm.) which calls for a large strong pocket--but it packs quite a wallop. Programs of up to 224 steps can be stored internally, or externally on magnetic cards.

To old-timers who grew up with wired calculators like the IBM 604, it is interesting to compare that machine (which, at one time, was doing most of the calculating work of this country) with the SR-52. The 604 had 80 wired program steps, generally 2-address (e.g., Read Out of a storage unit, Read In + to the accumulator). In contrast, the program steps of the SR-52 are something less than single address. For example, to move a number from storage unit 15 to storage unit 16:

```
RECALL
  1
  5
STORE
  1
  6
```

takes six program steps. On the other hand, the program steps of the 604 were limited arithmetically to add, subtract, multiply, and divide, whereas on the 52 a step can contain trig and log functions, or factorials, or any power of any number, or polar to rectangular coordinates. In addition, programming of the 52 includes subroutining, indirect addressing, and a full complement of jumps.

The 52 is still only a calculator; that is, it does not permit an instruction to operate on another instruction (the shining characteristic of computers). On balance, though, the 52's programming capability far exceeds that of the 604, except for speed.

The speed of the 52 is around 31 program steps per second for add-type instructions, or about 3 steps per second for the high-powered instructions (like cosine or arctangent). (The 604 executed add-type instructions at the rate of 2000 per second.)

The SR-52 operates on 12 decimal digit numbers and displays 10 digits either in fixed point or in scientific notation. Two levels of subroutining are built in, but further levels are possible through indirect addressing. Up to ten levels of parentheses can be used. Parentheses are needed, because of the algebraic logic of the machine. Integer arithmetic is awkward, since the machine has no INTEGER or MOD function; it is thus difficult to find the remainder after an integer division.

The beginner who is conditioned to computers will forget that the accumulator is also the display. Thus, the habit of writing

RECALL
1
5
SUBTRACT
7
JUMP ON MINUS

leads to trouble, since the operation "=" must be inserted before the jump.

The magnetic cards for recording programs are protected by affixing small tabs on the cards to enable the WRITE function; the tabs are then removed to inhibit re-recording. The tabs must be stuck to the cards very carefully or they tend to jam in the reader mechanism.

There is no indication in the display for a low battery condition. The usual indicators of a low battery are either wrong results (which may not be noticed) or the startup of the drive motor when no reading or writing has been called for.

The rating scale that POPULAR COMPUTING has used for desk calculators (see PC30-15) yields a score of 21.5 for the SR-52 at its current discounted price of $335.

The machine comes with 20 prerecorded programs (compound interest, random number generator, means and moments, factoring, three diagnostics, etc.) and 20 blank cards for recording user programs. The factoring program highlights the lack of a MOD function: to determine the primality of an integer, it takes $(3/2)\sqrt{N}$ seconds. Thus, to determine that 98939 is a prime takes nearly 8 minutes (and the program is claimed to work up to 10^{10}.)

The printer for the SR-52 is available, and is already discounted at $249.50. Control buttons for it are on the keyboard of the calculator.

Speaking of the manual: although written with unusual clarity and style, it still has the worst fault of all reference manuals; namely, it makes good sense only after you know what you're doing. Those things you want to know to get started are hard to find. The writer of the manual, having lived with the machine for months, has difficulty in putting himself in the position of the beginner. For example, the manual <u>does</u> indicate how to proceed to an arbitrary program step, but the average user will be a long time finding the reference.

The Hewlett-Packard 65 was out for two years before the SR-52 was announced. Since the technology is speeding up, it is reasonable to expect the 52 to be obsoleted before the end of 1976 (indeed, our annual Forecast predicts the appearance of the true pocket computer by then). Meanwhile, the SR-52 marks a significant advance in making computational power available to the individual. □

Problem Solution

Problem 52, Cube Route, PC16-1, called for calculating the final position in a trip in three dimensions. Starting at the origin, each of the 200 unit-length legs of the trip was dictated by three digits of the cube root of 16, taken as direction numbers. Thus, the first leg was to be in the direction given by the digits (2,5,1) as direction numbers (a,b,c). If we take

$$Q = \sqrt{a^2 + b^2 + c^2}$$

then

$$x_{new} = x_{old} + a/Q$$

$$y_{new} = y_{old} + b/Q$$

$$z_{new} = z_{old} + c/Q$$

for each leg of the trip. The first few legs, then, are given as:

leg number	direction numbers	end coordinates		
1	2,5,1	.3651483,	.9128709,	.1825741
2	9,8,4	1.074447,	1.543359,	.497818
3	2,0,9	1.291377,	1.543359,	1.474005

and so on. The difficult part of the calculation seemed to be the keypunching of 200 sets of 3 digits of the cube root of 16 (1001 digits of that cube root were given in issue 16). Many readers balked at the task of keypunching those 600 digits, although they would think nothing of writing and punching over 500 characters of a Fortran program to perform the calculation.

Associate Editor David Babcock and Publisher Gruenberger teamed up on this formidable task, with Gruenberger exercising his highest skill on doing the keypunching of the 600 digits of data and Babcock writing, keypunching, debugging, and testing the program. Their results for the terminus of the 200-leg trip are:

x = 97.69068856
y = 101.30865868
z = 100.01765382 □

CONTEST 7

K-LEVEL SIEVE

For our seventh contest, we have an extremely simple, but challenging problem. A great deal of work has already gone into its solution.

Start with the consecutive integers, from one up. At the first level of the procedure to be followed, select one number, reject the next, take the next, reject the next, and so on.

At the second level, operating on the numbers selected at the first level, we take the first two numbers, reject the next two, take two, reject two, and so on.

This process continues. At level K, operating on the numbers resulting from level K-1, we take the first K of them, reject the next K, take the next K, and so on.

WHAT NUMBERS WILL REMAIN?

Level one produces the odd numbers as its output.

Level two produces the sequence 1, 3, 9, 11, 17, 19, 25, 27,... or (8M-7, 8M-5) repeatedly.

Level three produces the sequence 1, 3, 9, 25, 27, 33, 49, 51,... or (24Q-23, 24Q-21, 24Q-15) repeatedly.

All sorts of algorithms suggest themselves for this problem. One could apply the definition of the problem directly: fill a block of storage with consecutive odd numbers and blank them out by twos; close up the gaps and blank out selected threes, and so on. Since each level cuts the number of remaining numbers in half, if one started this procedure with a million odd numbers, the process could continue only 17 stages and would yield just 17 numbers of the desired set. This is not the way to go.

Or, one could write subroutines to implement each level, with level one fed with consecutive integers. With 50 such subroutines, the final output of the 50th subroutine would yield 50 good terms of the sequence.

Or, since all these subroutines are identical in their logic, this method would lend itself to the use of one recursive subroutine.

All these schemes are fruitless in the sense of yielding very little output per minute of CPU time expended. [The recursive subroutine approach, implemented in Fortran on a CDC 3170, produced just the following:

1, 3, 9, 25, 57, 145, 337, 793, 1921, 3849, 8835, 18889

in a 5-minute run.]

The usual $25 prize will go to the person who produces, by computer, the longest output of the K-level Sieve problem. Each entry must include the program and a description of the algorithm used. All entries must be received by POPULAR COMPUTING (Box 272, Calabasas, California 91302) by July 31, 1976.

{A word of warning: the 39th term of the required sequence is a 14-digit number. Be prepared to use multiple-precision arithmetic on your algorithm.}

N-SERIES 38

Log 38	1.579783596616810156750072370481422344719319218856606
ln 38	3.637586159726385769426259553346030105312879395659384
$\sqrt{38}$	6.164414002968976450250192381454244225235624023444575
$\sqrt{38}$	3.361975406798963314840558566810563447406578969166259
$\sqrt[10]{38}$	1.438726886549915720849403839010265688372608516813068
$\sqrt[100]{38}$	1.037045558847565191194190852812715057064691206296665
e^{38}	31855931757113756.2203286717012986459995422099051810077532402988169852919049416088858588
π^{38}	7792829284694322855.623028408686094841050500379823889028297291490991018749324091515188
$\tan^{-1}$ 38	1.544486609541974427101412960678414222298833557211305

A Quick & Dirty Random Number Generator

Need a small quantity of random digits generated very simply? Simply divide Q/P and use the digits of the quotient.

Ideally, P should be fairly large and be a prime having 10 for a primitive root. In practice, it works nicely to pick a prime, P, for which (P-1)/2 is also a prime; these are plentiful. For example, one could use primes like:

2063	10463	200087
20663	32183	500519
66383	98663	1000667
98867	98927	2040287

The numerator, Q, can be any integer less than P; the choice will dictate where the sequence begins. The stream of digits produced would probably fail most of the standard tests of pseudo-random numbers (see the description of eight of these tests in PC33-7) but for many purposes would be acceptable. The words of caution by Don Knuth (see PC21-4) should be re-read before using this algorithm.

Given here is the reciprocal of 2063, carried to 1045 places. The reciprocal is full-sized; that is, it repeats on a cycle of 2062 digits. For such reciprocals, the second half is the 9's complement of the first half; the underlined portion shows this.

```
.00048473097430925836160930683470673776054289869122
 63693650024236548715462918080465341735336888027144
 93456131846825012118274357731459040232670867668444
 01357246728065923412506059137178865729520116335433
 83422200678623364032961706253029568589432864760058
 16771691711100339311682016480853126514784294716432
 38002908385845855550169655841008240426563257392147
 35821619001454192922927775084827920504120213281628
 69607367910809500727096461463887542413960252060106
 64081434803683955404750363548230731943771206980126
 03005332040717401841977702375181774115365971885603
 49006301502666020358700920988851187590887057682985
 94280174503150751333010179350460494425593795443528
 84149297140087251575375666505089675230247212796897
 72176442074648570043625787687833252544837615123606
 39844886088221037324285021812893843916626272418807
 56180319922443044110518662142510906446921958313136
 20940378090159961221522055259331071255453223460979
 15656810470189045079980610761027629665535627726611
 73048957828405235094522539990305380513814832767813
 86330586524478914202617547261269995152690 2569.....
```

Contest 3 Results

The contest in issue 32 (November 1975) was:

> A Fortran subroutine is to be written, to output all combinations of K things from an array of N things, one combination per call of the subroutine. The size of the array A (containing N things) is to be taken as less than or equal to 50. The subroutine, when called, is to put the next K of the N things into an array B. Assume that K is less than or equal to N. If the subroutine is called more than ${}_{N}C_{K}$ times, it is then to start over with the first combination.

with suitable conventions on the calling sequence of the subroutine.

When the deadline (February 29) was reached, all entries were passed on to our panel of judges. Every entry was carefully examined in what turned out to be a complex search for the most satisfying solution to the combinations problem. After struggling with the diverse styles and solution-methods of our contestants, the winner was chosen:

John Ferris
Hopewell, New Jersey

who receives our $25 prize.

Choosing between different excellent programs, all of which work, involves some subjective judgements. We are printing Mr. Ferris' program exactly as submitted, to show the workmanlike job that impressed the team of judges. The following comments about the program are by Mr. Ferris:

> As written, my routine prints a one-line echo of the parameters N and K. Depending on its use, it may be desirable to remove this write statement, but I felt it was safer to include it.
>
> Although not requested, I chose to make the routine respond to a fresh set of input parameters. This requires saving the input on first call and checking it on each successive call, but for this slight additional cost we have a routine that is much more useful, being able to handle any number of problems in one run. We also get some control over accidental changes in the input in the form of the echo message noted.
>
> The specifications called for real input and output arrays, but the routine need not operate under this restriction since it is only moving values around and not computing or converting.

For ease of checking, my test program uses as input the integers 1 to N.

A slightly faster version is possible by simply moving the B(J)= statement out of its separate DØ 62 loop and into the DØ 61 loop as

B(J)=ASAVE(IPØS)

thus assigning only those values that change. However, this is less defensive since it requires that vector B remain undisturbed between calls.

And the following comments came from the judges:

Ferris' routine handles all error conditions properly (including an attempt to use negative values for the parameters) and includes a simple and quite straightforward test routine. The program is written 100% in ANSI Fortran, as required by the contest rules. It is documented neatly and succinctly.

```
C   MAIN PROGRAM FOR TESTING SUBROUTINE COMB
      INTEGER LL(10),L(10)
      DO 5 J=1,10
    5 LL(J)=J
      DO 10 J=1,25
      CALL COMB(LL,L,6,3)
      WRITE (6,100) (L(JJ),JJ=1,3)
   10 CONTINUE
      DO 20 J=1,20
      CALL COMB(LL,L,6,2)
      WRITE (6,100) (L(JJ),JJ=1,2)
   20 CONTINUE
      DO 30 J=1,3
      CALL COMB(LL,L,10,10)
      WRITE (6,100) (L(JJ),JJ=1,10)
   30 CONTINUE
      DO 40 J=1,10
      CALL COMB(LL,L,5,1)
      WRITE (6,100) (L(JJ),JJ=1,1)
   40 CONTINUE
      DO 50 J=1,5
      CALL COMB(LL,L,1,1)
      WRITE (6,100) (L(JJ),JJ=1,1)
   50 CONTINUE
      CALL COMB(LL,L,51,0)
  100 FORMAT (1X,10I5)
      END
```

```
C  THE SUBROUTINE COMB
C----------------------------------------------------------------------------
C
C-------- PURPOSE ------------------------------------------------------------
C
C  THIS ROUTINE DETERMINES SUCCESSIVE COMBINATIONS OF N THINGS TAKEN
C  K AT A TIME.  EACH CALL GENERATES THE NEXT COMBINATION, WHICH IS
C  SELECTED FROM THE INPUT VECTOR A AND RETURNED IN THE OUTPUT VECTOR B.
C  AFTER ALL COMBINATIONS HAVE BEEN EXHAUSTED, THE NEXT CALL RETURNS
C  THE FIRST COMBINATION (SEE ALSO "ALGORITHM NOTES" BELOW).
C
C-------- PARAMETERS ---------------------------------------------------------
C
C  ON INPUT:
C     A = THE VECTOR FROM WHICH COMBINATIONS ARE SELECTED.
C     B = A VECTOR OF LENGTH K.
C     N = THE LENGTH OF VECTOR A (MAX=50).
C     K = THE LENGTH OF VECTOR B (MAX=N).
C
C  ON OUTPUT:
C     A = UNCHANGED.
C     B = THE NEXT COMBINATION OF VALUES FROM VECTOR A.
C     N = UNCHANGED.
C     K = UNCHANGED.
C
C-------- RESTRICTIONS -------------------------------------------------------
C
C  THE VECTORS A AND B MUST BE SINGLE WORD TYPE REAL, INTEGER OR
C  LOGICAL AND MAY NOT EXCEED A LENGTH OF 50.
C
C-------- ALGORITHM NOTES ----------------------------------------------------
C


C  THE FIRST COMBINATION RETURNED IS THE FIRST K VALUES FROM VECTOR A.
C  IN SUBSEQUENT COMBINATIONS, THE RIGHTMOST VALUE MIGRATES TO THE
C  RIGHT UNTIL IT REACHES THE NTH VALUE IN VECTOR A, WHEN THE NEXT
C  RIGHTMOST VALUE MOVES ONE TO THE RIGHT AND THE RIGHTMOST VALUE
C  STARTS ITS MIGRATION OVER AT THE NEXT VALUE TO THE RIGHT OF THAT.
C  THE LAST COMBINATION IS THE LAST K VALUES FROM VECTOR A, AND IF
C  THE ROUTINE IS CALLED AGAIN AFTER THAT WITH THE SAME INPUT, IT CYCLES
C  BACK TO THE FIRST COMBINATION AGAIN.  THUS THE USER MUST BEWARE
C  THE INFINITE LOOP AND MAKE AN APPROPRIATE INQUIRY IN THE CALLING
C  PROGRAM.
C
C  ANY CHANGE IN THE INPUT PARAMETERS A, N OR K WILL CAUSE THE ROUTINE
C  TO INITIATE A NEW SEQUENCE OF COMBINATIONS.  IF PARAMETERS N OR K
C  ARE OUT OF THE RANGE 1 TO 50, PROGRAM EXECUTION IS TERMINATED.
C
C-------- HISTORY ------------------------------------------------------------
C
C  THIS ROUTINE WAS CONCEIVED AND WRITTEN ENTIRELY BY JOHN FERRIS,
C  JANUARY 15, 1976, FOR ENTRY IN POPULAR COMPUTING COMPETITION #3.
C
C----------------------------------------------------------------------------
```

```
      SUBROUTINE COMB(A,B,N,K)
      DIMENSION A(N),B(K),ASAVE(50),LP(50)
      LOGICAL BADARG,FIRST
      DATA FIRST/.TRUE./
      IF (.NOT.FIRST) GO TO 40
C
C-------- FIRST CALL INITIALIZATION -------------------------------------
C
      FIRST=.FALSE.
      MAXLEN=50
C
C  CHECK RANGES ON ARGUMENTS N AND K
C
   10 BADARG=.FALSE.
      IF (N.GE.1 .AND. N.LE.MAXLEN) GO TO 11
           WRITE (6,600) N,MAXLEN
           BADARG=.TRUE.
   11 IF (K.GE.1 .AND. K.LE.50 .AND. K.LE.N) GO TO 12
           WRITE (6,601) K,N
           BADARG=.TRUE.
   12 IF (BADARG) STOP
      WRITE (6,602) N,K
C
C  SAVE INPUT
C
      DO 21 J=1,N
   21 ASAVE(J)=A(J)
      NSAVE=N
      KSAVE=K
C
C  INITIALIZE OUTPUT AND LIST OF POSITIONS SELECTED, LP
C
   30 DO 31 J=1,K
      B(J)=ASAVE(J)
   31 LP(J)=J
      GO TO 999
C
C-------- CHECK INPUT:  IF IT HAS CHANGED, ASSUME A FIRST CALL ---------
C
   40 IF (N.NE.NSAVE .OR. K.NE.KSAVE) GO TO 10


      DO 41 J=1,N
      IF (A(J).NE.ASAVE(J)) GO TO 10
   41 CONTINUE
C
C-------- GENERATE NEXT COMBINATION -------------------------------------
C
      LOOK=K
      MAXPOS=N
C
C  LOOK THROUGH LIST LP FOR NEXT POSITION TO UPDATE
C
   50 IF (LP(LOOK).LT.MAXPOS) GO TO 60
      LOOK=LOOK-1
      IF (LOOK.EQ.0) GO TO 30
      MAXPOS=MAXPOS-1
      GO TO 50
```

```
C
C   UPDATE LIST LP AND CONSTRUCT OUTPUT VECTOR B
C
   60 IPOS=LP(LOOK)+1
      DO 61 J=LOOK,K
      LP(J)=IPOS
   61 IPOS=IPOS+1
      DO 62 J=1,K
      JP=LP(J)
   62 B(J)=ASAVE(JP)
  999 RETURN
C
C-------- FORMAT STATEMENTS ------------------------------------------------
C
  600 FORMAT (54H0ON A CALL TO COMB, ARRAY A WAS GIVEN AS HAVING LENGTH,
     &          4H N =,I5/5X,26HCORRECT RANGE IS FROM 1 TO,I4)
  601 FORMAT (54H0ON A CALL TO COMB, THE NUMBER OF THINGS TO BE SELECTE,
     &          31HD FROM ARRAY A WAS GIVEN AS K =,I5/5X,13HCORRECT RANGE,
     &          54H IS FROM 1 TO A VALID VALUE OF N (WHICH ON THIS CALL W,
     &          2HAS,I4,1H))
  602 FORMAT (16H0COMBINATIONS OF,I3,13H THINGS TAKEN,I3,10H AT A TIME).
      END
```

Problem Definition

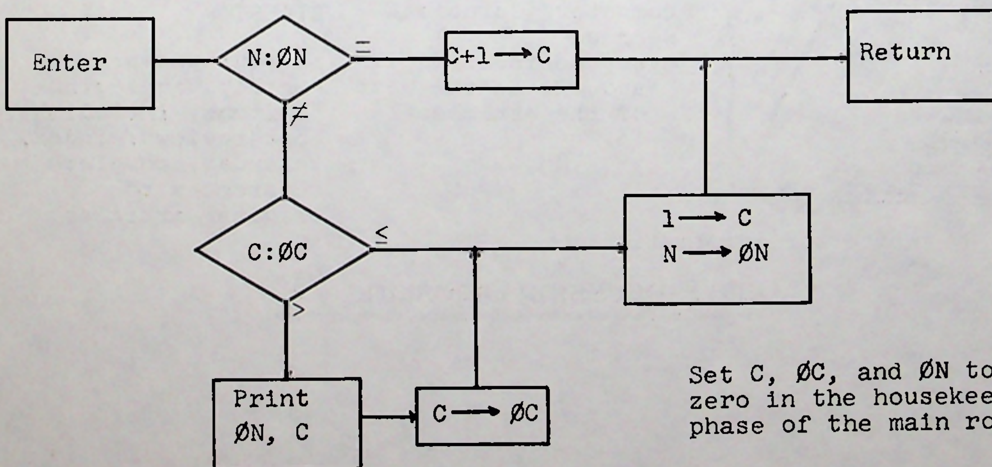
Every treatise on problem solving points out that the initial stages of the process are some form of:

1. Be aware that a problem exists.
2. Define the problem carefully.

Step one is usually completed when someone states a problem. Consider our Problem 117:

> A subroutine in a program produces as its output 3-digit numbers (N) in no special order and with possible strings of duplicates. A second subroutine is needed which will examine these numbers as they are produced and report each new longer string of duplicates.

The problem was defined in PC34-18 by a long example, and a flowchart for a solution was sought. Such a flowchart was given in PC35-17. It contained an egregious error (inversion of the inequality signs on one of the decision boxes) and the corrected version is given here:

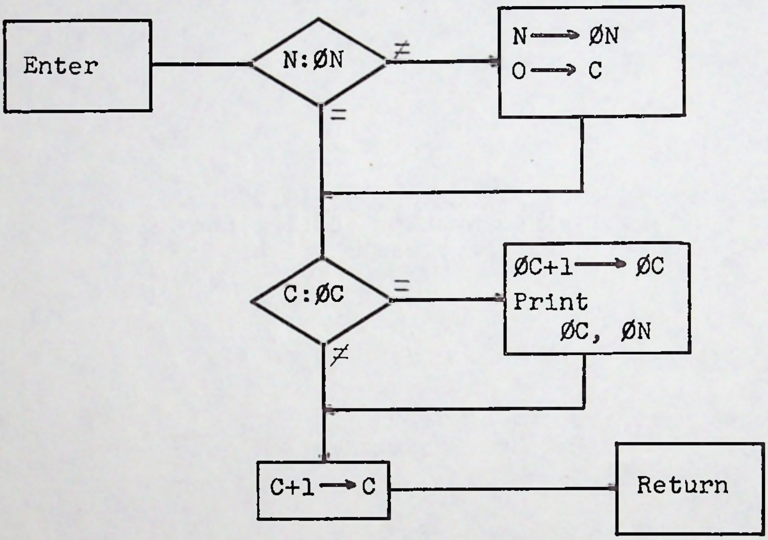


Set C, ØC, and ØN to zero in the housekeeping phase of the main routine.

This error was pointed out by Richard D. Smith, Princeton, New Jersey, who added:

> The sequence 1, 2, 2, 2, 3,... would not report the first occurrence of a string of length 2. Printing should occur when N = ØN.
>
> Due to faulty initialization for the given logic, an infinite string of zeros would print nothing.

Mr. Smith presented the following flowchart:



Set C and ØC to zero in the housekeeping phase of the main routine.

and the following observations:

> Always checking C against old C is redundant except to pick up the first sequence of one.
>
> Having ØC incremented by one instead of having C+1 replace ØC appears at first glance to be unstable because two counters (C and ØC) are running in parallel. However, in this case destruction of the C value will be corrected on the next new N and destruction of ØC will merely repeat or skip a series--this being obvious since the ØC printed is always the entire set of integers. The case where C is greater than ØC is ignored.

The problem originally arose in the calculation of string lengths in the 3X+1 problem (see PC13-12). It had been observed that all values of N from 596310 through 596349 finish the 3X+1 process with 98 terms--a record string of 40 successive integers. What was being sought in Problem 117 was the logic for detecting ever-longer strings. Thus, the appearance, in the example just cited, of another string of 40 was to be ignored.

And thus, the flowchart (as corrected) given was correct for the problem we had in mind, and Mr. Smith's flowchart is correct for a different interpretation of the problem. Specifically, for the sequence 1,2,2,2,3,4... we wanted output of

```
1   1
3   2
```

whereas his logic would produce

```
1   1
2   2
3   2
```

Or, the example given of a string 0,0,0,0,0,0,0,0... in our logic would not print anything until a non-zero value appeared, whereas his logic would print

```
1   0
2   0
3   0
4   0  etc.
```

All of this comes under the heading of "Let's state our problems clearly, huh?" Mr. Smith's comments point up a basic concept in computing.

You are sitting in the airport waiting room, with nothing to do but play with your new pocket calculator. It is a fine machine, good to 10 significant digits for all its functions, which include square root, logarithm, and trigonometric functions. It also has an additive storage unit.

So you sum, for the values from 1 to 10: the integers; their square roots; their natural logarithms; and their sines, cosines, and tangents (with the arguments in radians).

What will be the final result?

[Using an SR-50, we get 83.55033448246]

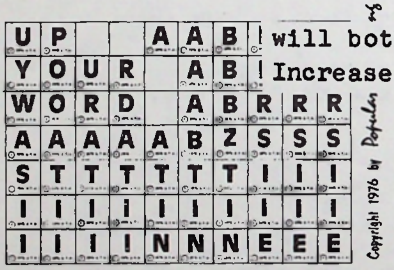
M	O	N	O	P	O	L			?
	C	R	A	B	B	L	E		?
M	A	S	T	E	R	M	I		?

...balderdas !

IT'S TIME TO TRY *Up Your Word*

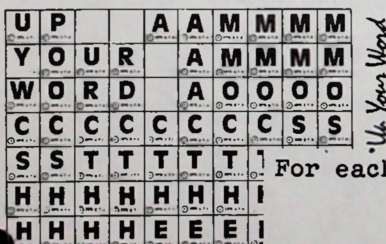
THE NEWEST, MOST CHALLENGING WORD GAME

For two or more players.



_______ will both challenge and entertain you and your friends. Increase your word skill--organize a tournament!

The complete game, along with instructions and several variations, is only $2.00!



ORDER FROM *Popular Computing*

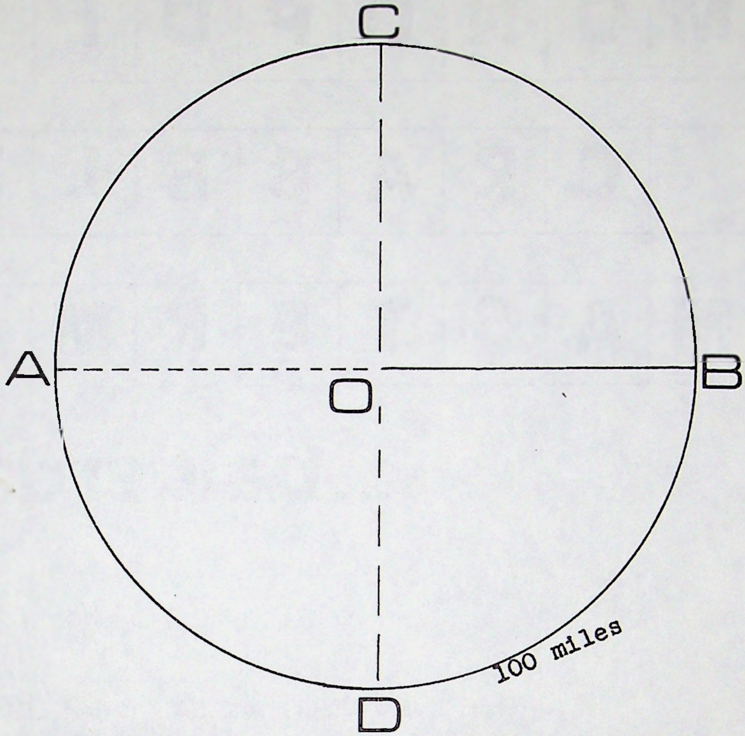
BOX 272 CALABASAS, CA. 91302

For each game, send $2 plus 30¢ for postage and handling (U.S., Canada, and Mexico)
or plus 50¢ for postage and handling (all other countries)

For two games, send $4 plus 50¢ (U.S., Canada, and Mexico)
or plus 75¢ (all other countries)

California residents add 12¢ per game.

A Circuitous Race



Given a circular road of length 100 miles. A car starts at O, goes to B, and travels the circle counter-clockwise, back to B. It then crosses the circle from B to A, after which it again circles the road. This procedure (a complete circle, followed by a crossing on the diameter) is repeated for twenty complete revolutions, after which the car returns to the center. The distance travelled is then:

$$(2000 + 2000/\pi) = 2636.61978 \text{ miles.}$$

There are four such cars. No. 2 starts at O and heads North (from O to C); No. 3 starts West (from O to A); No. 4 starts South (from O to D). The four cars all start at the same time, and each makes a trip the same way. Their speeds are:

$$\left.\begin{array}{ll} \text{No. 1} & 67 \\ \text{No. 2} & 29 \\ \text{No. 3} & 59 \\ \text{No. 4} & 41 \end{array}\right\} \text{ miles per hour}$$

All four cars eventually return to O, and all have travelled the same distance.

The Problem is: How many times has any car overtaken and passed another car?